

Development of a Car-like Online Navigation Testbed

Kito Berg-Taylor Keehong Seo Soon-Jo Chung

Department of Aerospace engineering, Iowa State University, Ames, Iowa 50011

Email: {kito, kseo, sjchung}@iastate.edu

Abstract—We present new realtime path planning and collision avoidance algorithms for an autonomous rover equipped with a laser range finder to be used as a platform for multi-agent navigation and control in unknown environments. For successful navigation, such tasks as localization, map-building, and collision avoidance should be handled at the vehicle level. The proposed architecture covers these aspects of robotic path-planning in a modular and robust manner, allowing quicker development of more sophisticated path-planners. Using a conventional SLAM algorithm, a feature map and the location of the vehicle is obtained. The information for orientation and distance of the obstacles ahead is available from a laser range finder. The proposed collision avoidance algorithm provides multiple paths to guide the vehicle through the environment. The system acts as a self-contained extendable platform for development and testing of high-level pathfinders.

I. INTRODUCTION

This paper presents a unified platform for development of multi-vehicle path-planning and path-following behaviors in an unknown environment. The platform encompasses a highly modular combined hardware and software approach to the development of a robust testbed for research and testing. The modularity of the platform also allows it to be seamlessly integrated into distributed multi-vehicle systems. In particular, emphasis is placed on developing new realtime collision-avoidance path planning algorithms. Several methods are proposed and discussed.

II. PLATFORM

A. Hardware

The vehicle test platforms are a series of modified Traxxas E-Maxx RC trucks. The vehicle's body has been replaced with a navigation system comprising a 1.6 Ghz mini-ITX computer, 5Ah lithium polymer battery, 802.11G wireless card, and a 2-axis magnetometer. The box has a SICK S300 scanning laser rangefinder attached to it, which communicates with the navigation computer over RS-422. The laser rangefinder provides distance measurements to obstacles within 25 meters in the surrounding 270 degrees in one-half degree increments. The measurements are provided at 80 Hz and reach to a distance of 30 meters.

B. Software Architecture

The software architecture is designed as a modular message-based system. The software is broken down into individual

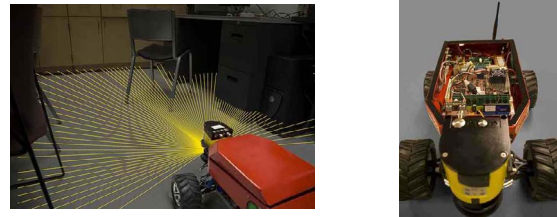


Fig. 1. The vehicle test platform as equipped with a scanning laser rangefinder

modules each of which implements a particular feature, such as interfacing with the laser rangefinder or planning an obstacle free path.

The inputs and outputs to the module are shared through a message-passing architecture. Using this system we are able to fully exploit the parallel distributed nature of the system. Each message is system independent and can be safely passed from one machine to another. In this way several testbeds can share information for collaborative tasks as well as relaying data back to a ground station. For transmission over the wireless network messages are serialized by the sender and deserialized at the receiving end.

III. ENVIRONMENT MAPPING

A primary prerequisite of any robust robotic navigation system is an up-to-date and accurate map of the environment. Generally errors in the vehicle's estimate of its position necessitate updating the vehicle's location by triangulating landmarks in its surroundings. However, the location of these very landmarks depends on the location of the vehicle itself. To this end it is necessary to use a form of Simultaneous Localization and Mapping (SLAM) to build a map of the environment while also localizing the vehicle against this map. SLAM is one of the more difficult problems in robotics [1], but several effective methods have been studied in this area.

On this testbed a map of the environment is generated using the FastSLAM method presented in [2]. Its strength lies in its ability to maintain multiple hypothesis on the environment while processing large maps online. This method has been previously shown to scale well to maps with upwards of 50,000 recorded features. FastSLAM is the primary method

of localizing the vehicle as it drives around in an unknown environment.

IV. OBSTACLE AVOIDANCE

We introduce two new path planning and obstacle avoidance algorithms in this section. Both are found to be useful in exploring indoor environments. The approximated circular sector expansion algorithm has its advantage in economical computation time. Once line features are extracted from the map by using our line feature detection algorithm in [3], our new path planning algorithm can generate multiple paths as a binary tree format. The optimization of the path is also discussed.

A. Method I: Approximated Circular Sector Expansion

As a general pathfinding and navigation testbed the vehicle must be capable of dealing with unexpected obstacles in its general vicinity as its low-level behavior. A robust yet efficient method of detecting and avoiding obstacles was developed to keep the vehicle safe while heading to its target. A particular emphasis was placed on efficiency to reserve as many processor cycles as possible for future use.

Here we present a heuristic approximation to the circular sector expansion (CSE) method presented in Ref. [4]. This method provides an approximation of the generalized Voronoi diagram (GVD) of arbitrary set of points arranged about a starting location. The output of the method will provide a path which is equidistant from obstacles on both sides of the path, as well as a “safe radius” circle within which there are no obstacles. For multi-vehicle path planning, this method may be used by finding a trajectory which attempts to keep every vehicle in a swarm evenly distributed within these safe circles.

Our approximation is shown to be accurate in the majority of real-world cases with most expanded circles devoid of obstacles in their interior. While it presents at best a constant time improvement over the $O(n)$ complexity of the original CSE, it allows for simple optimizations as well as expansion against non-point obstacles. This allows for its usage in structured environments with walls represented as lines, as well as being robust in unstructured environments. When combined with a feature extraction routine it is expected to gain significant speed improvements over the original CSE algorithm for large data sets.

1) *Initial Conditions:* CSE operates directly on the output of a bearing and range sensor such as a scanning laser rangefinder. It requires only the location and heading of the vehicle and a representation of the obstacles present locally.

The starting circle is centered on the vehicle and expands to be tangent to the closest obstacle. We will represent this with a moving “expansion point” $c \rightarrow \mathbb{R}^2$. Each circle will be expanded from this point sequentially. The center of the starting circle is located at

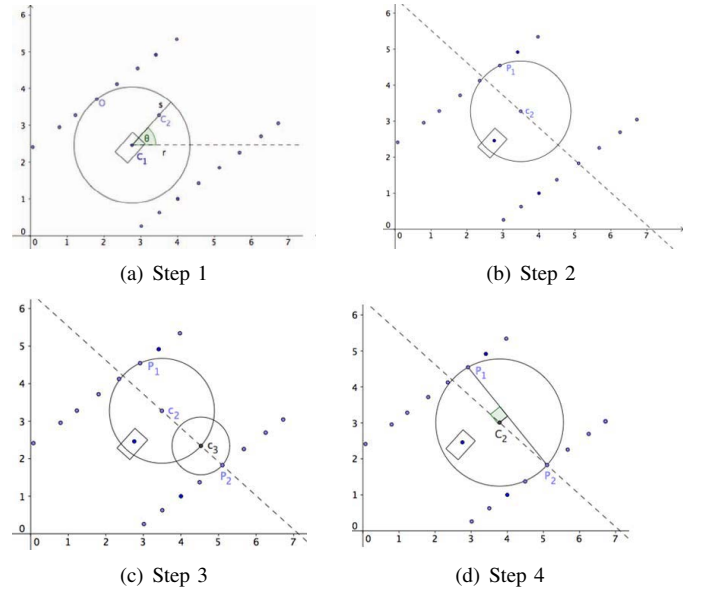


Fig. 2. The four-step circle expansion process starting from the vehicle location.

$$C_1 = x_k \quad (1)$$

where x_k is the vehicle's location at the current time k . This circle has a radius of

$$r_1 = \min_{i \in [0, n]} (d(x_k, x_k + s_i(\cos \alpha_i, \sin \alpha_i)^T)) \quad (2)$$

where s_i is the straight-line distance to the i^{th} obstacle and α_i is the heading to the same obstacle. The first expansion will be perpendicular to the forward direction of the vehicle, at a distance proportional to the radius of the first circle.

$$c_i = c_1 + (r_{i-1} - r_{min})[\cos \theta, \sin \theta]^T \quad (3)$$

r_{min} is a predefined minimum safe radius, usually taken as the radius of the smallest circle that would fully contain the vehicle, and θ is the vehicle's heading.

The first expansion's forward direction is taken to be that of the vehicle.

$$\vec{f}_1 = [\cos \theta, \sin \theta]^T \quad (4)$$

2) *Circle Expansion:* After the location of the expansion point is found, the first circle expansion can begin. Figure 2 shows the sequence of expansions starting from the vehicle center through to the first fully expanded circle. The circle's center is constrained to a line perpendicular to the expansion forward direction.

$$\hat{l} = [c_i + \lambda \vec{f}_{i-1}^\perp \mid \lambda \in \mathbb{R}] \quad (5)$$

The closest point to the expansion point is defined as

$$P_1 \equiv \min_{i \in [0, n]} (d(c_i, c_i + s_i(\cos \alpha_i, \sin \alpha_i)^T)) \quad (6)$$

The expansion point is then relocated to the point lying on both the circle and the line in the direction away from the obstacle.

$$c_{i+1} = c_i + \{(c_i - P_1) \cdot \hat{l}\} \hat{l} \quad (7)$$

This expansion sequence is repeated iteratively until the closest obstacle to the expansion point c_i is not P_1 . The obstacle that causes this condition is then defined as P_2 . The center of the fully expanded circle can then be found as

$$C_{i+1} = (\overline{P_2 P_1})^\perp \cap \hat{l} \quad (8)$$

and the radius is

$$r_{i+1} = \|C_{i+1} - P_1\| \quad (9)$$

The expansion is then repeated until some termination condition [3] is reached.

3) *Forward Direction*: After a circle has been expanded it is necessary to pick a new forward direction to continue the expansion. The logical choice for a “safest-path” expansion is to expand away from both previously detected obstacles P_1 and P_2 . The safest-path expansion is defined as

$$\vec{f}_{i+1} = \{\vec{f}_i \cdot (P_2 - P_1)^\perp\} (P_2 - P_1)^\perp \quad (10)$$

To take the vehicle dynamics into account, the rate of change of the forward direction can be limited.

B. Alternate Obstacle Representations

When operating vehicles in a structured indoor environment it can be very advantageous to represent obstacles as lines instead of a series of points. The presented method is capable of generating the GVD of environments that are composed of both lines and points without requiring the use of a grid-based method as presented in [5]. This is done by calculating the perpendicular distance to each line in addition to the point obstacles. The addition of line obstacles improves the accuracy of this method with no additional cost above that necessary to find the lines. The detailed description for detecting line features is found in [3]

C. Method II: Triangle Edge Cutting

Suppose that, after processing raw data from range finder with a feature detection algorithm, we have a map of obstacles that are represented not only by points but also by lines. Such representation is computationally economical in structured indoor environments. To fully benefit from it, we propose a new path planning algorithm that is suitable for a rover navigating through indoor environments.

1) *Basic Idea*: The proposed algorithm performs an incremental triangulation over the points on the map, which could be end points of lines or nonlinear obstacles of arbitrary shape. Since each triangle is potentially a branching point of the path, the triangulation result is stored as a binary tree. From the binary tree, multiple paths can be identified by cutting through the edges of the triangles. In fact, given a triangulation of points on a 2D map, one can start from any edge to advance either to the left or to the right edge repeatedly unless it is part of a line obstacle or shorter than a threshold. From every edge, the next triangle is selected by recruiting a new point in front of the edge, which is done by checking the closest point from the edge. Figure 4 (a) shows an example of triangulation, which shows the overlay of all the possible triangulations from a fixed starting edge. In this way, one can find paths that avoid obstacles on the map whether they are static or moving. A simple way for the vehicle to use the result of triangulation as its path is referring the vehicle to way points on the path. The crossing of a path with an edge is used as a way point for the vehicle. We also show that, given a cost function to be minimized, the way point position on each edge can be optimized.

2) *Special Cases*: In addition to the basic description of the algorithm, there are certain cases when we need to insert an edge to the tree before advancing to the next triangle. Suppose we are currently at edge E_0 and about to advance to E_1 in Fig.3 if the foot A of the perpendicular line h from P_1 toward E_2 is found on E_2 , then an additional edge will be inserted as a node in the tree before E_1 . The edge to be inserted will be the shortest of the line segments between P_1 and the following points: A (if E_2 is a wall), B , or C . Point C is the intersection of h and a equidistant curve S from E_0 through P_{new} . If there is a line obstacle L_1 between C and P_1 , then B should be chosen for the edge to be inserted instead of C . This action improves clearance of the path away from P_1 .

3) *Properties*: One can determine if each path is terminated as a dead end, an open end, or a loop. If both E_1 and E_2 are walls, or E_0 is open but shorter than a threshold width, then it is supposed as a dead end. If there is no further point to recruit from the current edge, then it is regarded as an open end. If an edge was previously visited and if the direction of advance also is the same as the previous visit, then it is regarded as a loop.

The terminating condition can be used to narrow down the choice of the paths appropriate for the application. If the application requires the vehicle to proceed without stopping at blocked corners or without circling around an object, then it is obvious that the open-end paths must be chosen. This selection can substantially reduce the number of possible paths. From the open-end paths chosen, one can either randomly choose a path or, if target position or direction is known, deliberately choose a path that leads to the target possibly with the shortest distance to travel. Again depending on the purpose of application, one may want to choose a path so that the sum of the areas of the visited triangles is the largest if the mission is to cover a certain area (i.e., coverage problem).

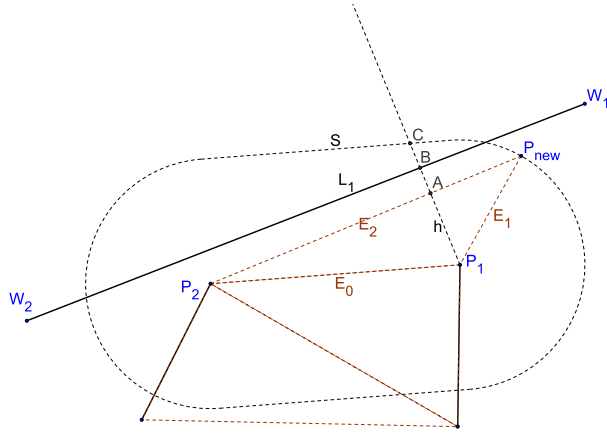


Fig. 3. When the base angle of the triangle at P_1 is large enough for A to exist on E_2 , additional edge will be inserted between E_0 and E_1 . The edge will be the shortest of the line segments between P_1 and A (if E_2 is a wall), B , or C . This action helps ensuring the clearance of the path from obstacles by checking the length of the edge inserted.

Once a reasonable number of paths are chosen, one can perform optimization of the way point position on each edge, for example, to minimize overall curvature.

A singularity of the algorithm is that when there is no point or line at all in the map, then it can not find a point to draw a triangle, and thus cannot find any path even though it is facing a large open space. This issue can be solved when generating a map from any sensor data. For example, one can place points at virtual boundaries when the scan rays do not hit any objects.

4) *Simulation*: To demonstrate the feasibility of the algorithm, we used actual sensor data stored from the laser range finder of the rover moving in a hallway. After rendering the data with line detection algorithm, we also added imaginary clustered points to represent obstacles. Fig.4 shows the map generated from one scan by the range finder and the paths with different terminating conditions as open end, dead end, and loop. Green lines mark the edge where the triangulation initiated on which the vehicle was actually located. The red x marks in Fig.4(a) represent mid points of the edges to travel through. In Fig.4(b) to (d), the solid lines are drawn by connecting way points on edges and it can be used as a reference path for the vehicle.

The way points are optimized with the following cost function with the gradient-descent method.

$$J = \sum_{i=2}^{n-1} (\cos(W_{i-1}W_iW_{i+1}) + 1), \quad (11)$$

where n is the number of way points W_i . The location of the starting point is fixed as a random point on the first edge and the last way point is fixed to be the mid point on the last edge. The optimization will try to enforce the angles to be close to 180 degrees to make the overall path as straight as possible to save the effort for turning. This allows the vehicle to steer away from the corner before reaching it, which is usually a

better way than to drive straight toward the corner and make a stiff turn. When the cost function is more complicated, genetic algorithm can be considered for use. The optimization result can be used to verify if the path will be feasible given the minimum turning radius in advance. If the minimum turning radius is a function of linear velocity, the forward speed can be adjusted based on our on-line look-ahead path planning.

5) *Pseudo Code*: The pseudo code for the triangulation algorithm is given in Algorithm 1.

Algorithm 1 Triangulation Binary Tree

```

(t, T) = tri( $P_1, P_2, T, W, V$ )
// $P_1, P_2$ : right and left end points of the current edge  $E_0$ 
// $W$ : walls,  $V$ : visited edge
// $N_{fwd}$  = Number of points in the front (visible or not)
if Visited( $P_1, P_2, V$ ) then
    Loop Detected
    Store Current Edge
else if  $E_0 < E_{min}$  or  $E_0 \in W$  then
    t = DeadEnd
else
     $P_{new}$  = closest and visible point in the front)
    if ! $P_{new}$  and  $N_{fwd} > 0$  then
         $t_l$  = DeadEnd,  $t_r$  = DeadEnd
    else if !( $P_{new}$ ) and  $N_{fwd} = 0$  then
         $t_l$  = OpenEnd,  $t_r$  = OpenEnd
    else
        Add  $E_0$  to  $V$ 
        Find  $E_1^*$  to be inserted before  $E_1$ 
        if Need  $E_1^*$  and  $E_1^* < E_{min}$  then
             $t_r$  = DeadEnd
        else
            ( $t_r, T$ ) = tri( $P_1, P_{new}, T, W, V$ )
            if Need  $E_1^*$  then
                Append  $P_1P_2^*$  to  $T$ 
            end if
        end if
        Find  $E_2^*$  to be inserted before  $E_2$ 
        if Need  $E_2^*$  and  $E_2^* < E_{min}$  then
             $t_l$  = DeadEnd
        else
            ( $t_l, T$ ) = tri( $P_{new}, P_2, T, W, V$ )
            if Need  $E_2^*$  then
                Append  $P_1^*P_2$  to  $T$ 
            end if
        end if
        end if
        Append  $E_0$  to  $T$ 
    end if

```

6) *Consideration for On-line Implementation*: Computation time is a critical factor in using the algorithm for real-time applications. General efforts to reduce computation time can be considered. Since the visibility is checked in the algorithm when recruiting a new point from the current edge, using the sweep line algorithm [6] can reduce the computational effort

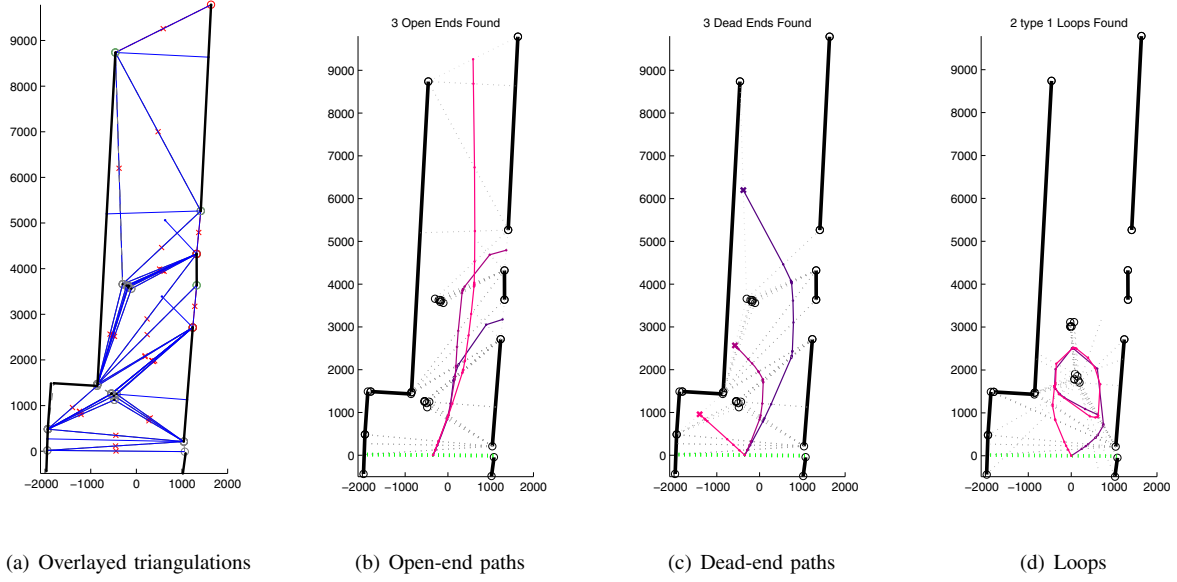


Fig. 4. The algorithm generated multiple paths, categorized them with ending conditions, and optimized the way point locations. Green lines are the starting edges on which a vehicle was sitting. Imaginary points are added to represent obstacles marked as black circles. (a) Overlaid triangulations for all the branches in the binary tree (b-c) Paths that end with open or closed space (d) Looping paths were found when the obstacles were moved.

from $\mathcal{O}(n^2)$ to $\mathcal{O}(n \log n)$. Also, if the map purely consists of lines without clusters of points, it performs best. If one is using SLAM when navigating the vehicle, registering the triangles as objects in SLAM method can also save much resource because the triangulation can be done incrementally at each frame without starting from scratch.

7) *Extension to Three Dimensions:* The algorithm can be extended to a 3 dimensional sensor data. Since the acquisition of such 3 dimensional data is available from the literature [7], all we need is to replace a triangle with a tetrahedron, in which case we have a trinary tree. Now, an edge in 2D is translated to a side of a tetrahedron, or a triangle. If a way point on an edge was parameterized in 1 dimensional variable, now it is 2 dimensional. A merit of the proposed algorithm in extending to 3D is that the algorithm requires not all the points in the 3D space but only vertices, lines and walls to be identified, hence data gathered from a vision sensor can be exploited. The application includes autonomous indoor flight of micro aerial vehicles. Fig. 5 visualizes the concept.

The extension of the proposed algorithms could be comparable with [8]. The differences are that our method do not depend on the grids, and that both the circle expansion and the triangulation are performed incrementally hence no need to search for connected spheres.

V. GUIDANCE COMMAND

We introduce a guidance control law that can efficiently make use of the safe paths generated from the path planning algorithms described in the previous section.

$$\tan \phi = -\frac{2L}{r_1} \sin \theta_1, \quad (12)$$

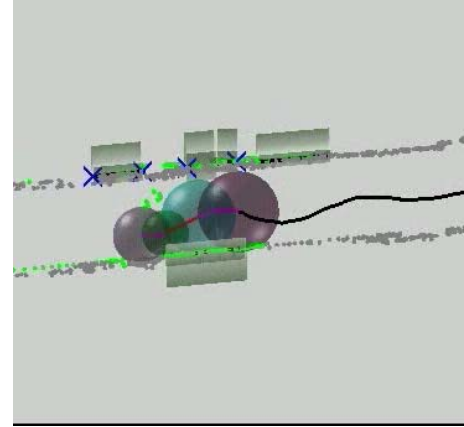


Fig. 5. 3 Dimensional extension of the method I, approximated CSE

where ϕ is the steering angle of the front wheel, L is the distance between front and rear wheels, r_1 is the distance from the vehicle to the center of the first circle (if method I) or first way point (if method II), and θ_1 is the initial heading error. The stability analysis for similar guidance laws with limited assumptions can be found in [9], [10].

If we incorporate the second way point in the control law, the new guidance command is written as

$$\tan \phi = -\frac{2L}{d} \sin(\theta_1 + \alpha), \quad (13)$$

where

$$\begin{aligned} d^2 &= r_1^2 + r_2^2 + 2r_1r_2 \cos \theta_2 \\ \cos \alpha &= \frac{r_1^2 + d^2 - r_2^2}{2r_1d} \end{aligned} \quad (14)$$



Fig. 6. A time-lapse photo showing the vehicle navigating around static obstacles

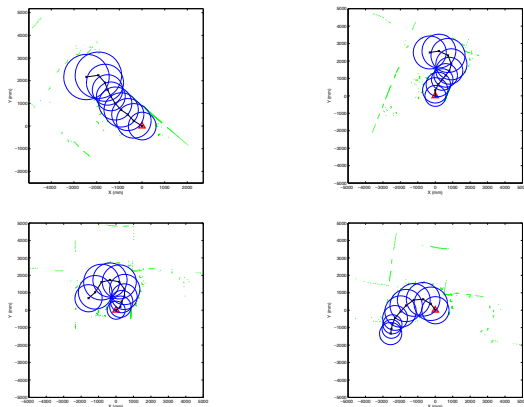


Fig. 7. A series of circle expansions on a pre-recorded drive segment

VI. DEMONSTRATION RESULTS

The vehicle has shown itself to be robust enough to operate on ice, in mud and in complex indoor environments. However it has been observed that the vehicle's control can be dependent on several outside factors including temperature and battery charge level. The implementation of hall-effect odometry sensors and a closed-loop speed controller is expected to improve the consistency of the vehicle.

Figure 7 shows a sequence of circle sector expansions as calculated on a pre-recorded drive segment. The vehicle was placed in a course which had been setup at the intersection of two hallways. It then drove around the oval track for several laps while collecting data from its laser rangefinder. This data was then used for offline testing of the CSE algorithm in MATLAB. The figure shows that the algorithm was quite capable in the area of short-term planning. Within the limits of the available data the algorithm did an excellent job of locating a feasible path.

The expansion sequence was limited to ten circles because it was observed that extending the sequence past that provided little in terms of useful additional results. With this limit in place the code ran at approximately 30Hz against 540 obstacle

points which were reduced before every iteration to eliminate excessively close points.

The obstacle avoidance algorithm has been evaluated during several runs through the hallways of the Howe aerospace engineering building against both static and dynamic obstacles. Figure 6 shows the vehicle successfully navigating a corner with two obstacles present.

VII. CONCLUSION

We proposed new path planning algorithms that are suitable for a rover navigating through indoor environments. In particular, the proposed Triangle Cutting Edge method is computationally efficient in structured indoor environments that can be represented not only by points but also by lines. Results of simulations and experiments show the effectiveness of the proposed approaches. We also presented a robust networkable platform for the development of both single-vehicle and multi-vehicle pathfinding algorithms. The platform implements a modular software architecture which uses a message-passing system to share information both among modules and across the network. The platform also takes care of tasks critical to any pathfinding experiment, including generating a map of the environment and performing obstacle avoidance and vehicle control activities. This platform is expected to be a valuable time-saver for the development and analysis of future pathfinding and path-following behaviors.

ACKNOWLEDGMENTS

This research was partially supported by the Air Force Office of Scientific Research (AFOSR) and the Information Infrastructure Institute (iCUBE) at Iowa State University. The authors are also grateful to Drs. Arun Somani and James Oliver for their support.

REFERENCES

- [1] S. Thrun and W. Burgard, *Probabilistic Robotics*. MIT Press, 2005.
- [2] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit, "FastSLAM: A factored solution to the simultaneous localization and mapping problem," in *Proceedings of the AAAI National Conference on Artificial Intelligence*. Edmonton, Canada: AAAI, 2002.
- [3] K. Berg-Taylor, "Localization and obstacle avoidance for small agile vehicles," in *IEEE Region V Conference*, Apr 2008, p. to appear.
- [4] S. Ronnback, T. Berglund, and H. Freriksson, "On-line exploration by circle sector expansion," in *Proceedings of the IEEE Conference on Robotics and Systems*. Kunming, China: IEEE, 2006.
- [5] H. Choset, K. M. Lynch, and S. Hutchinson, *Principles of Robot Motion*. MIT Press, 2005.
- [6] M. Shamos and D. Hoey, "Geometric intersection problems," in *Proc. 17th Annu. IEEE Symp. Found. Computer Sci.*, 1976, pp. 208–215.
- [7] S. Thrun, W. Burgard, and D. Fox, "A real-time algorithm for mobile robot mapping with applications to multi-robot and 3d mapping," in *IEEE International Conference on Robotics and Automation*, Apr 2000.
- [8] J. A. O. Vandapel, N.; Kuffner, "Planning 3-d path networks in unstructured environments," in *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, vol. 18, no. 22, Apr 2005, pp. 4624–4629.
- [9] A. Ollero and G. Heredia, "Stability analysis of mobile robot path tracking," in *Proceedings of the Institute of Electrical and Electronics Engineers/Robotics Society of Japan International Conference on Intelligent Robots and Systems*. IEEE, Piscataway, NJ, 1995, pp. 461–466.
- [10] S. Park, J. Deyst, and J. P. How, "Performance and lyapunov stability of a nonlinear path-following guidance method," *JOURNAL OF GUIDANCE, CONTROL, AND DYNAMICS*, vol. 30, no. 6, pp. 1718–1728, Dec 2007.